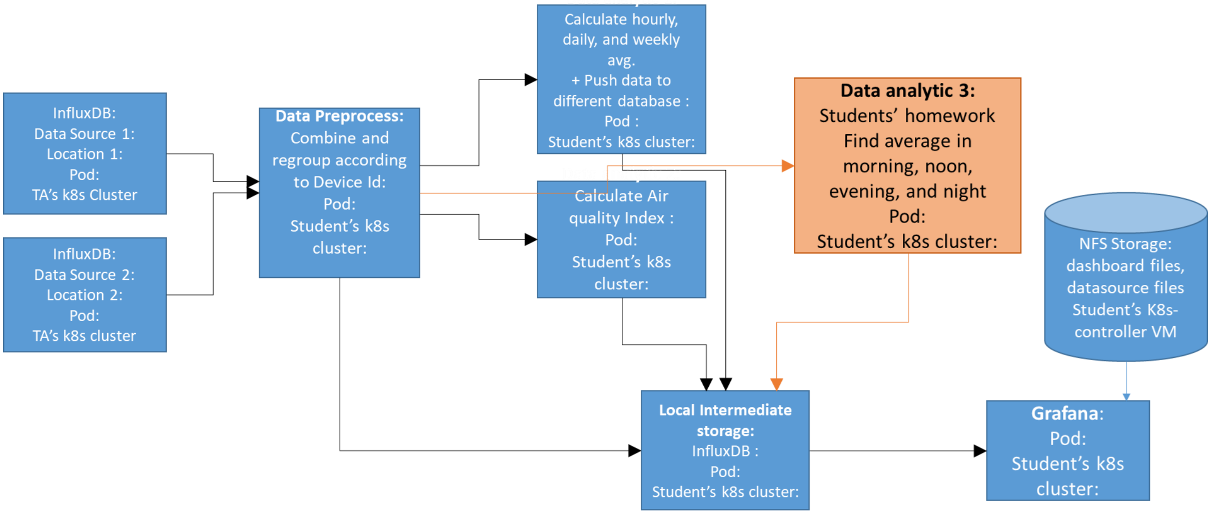
Practice Session-11:- **Working with data intensive application**

This practice session mainly focuses on the small demonstration of dataops, where multiple stakeholders such as data analyst, data scientist, developer and operation engineers are involved in rapid development of data-intensive/big data applications.

The following scenario is used in this practice session.


InfluxDB:
Data Source 1:
Location 1:
Pod:
TA's k8s Cluster
InfluxDB:
Data Source 2:
Location 2:
Pod:
TA's k8s cluster
Data Preprocess: Combine and regroup according to Device Id: Pod: Student's k8s cluster:
Calculate hourly, daily, and weekly avg. + Push data to different database : Pod : Student's k8s cluster:
Calculate Air quality Index : Pod: Student's k8s cluster:
Data analytic 3: Students' homework Find average in morning, noon, evening, and night Pod: Student's k8s cluster:
NFS Storage: dashboard files, datasource files Student's K8s-controller VM
Local Intermediate storage: InfluxDB : Pod: Student's k8s cluster:
Grafana: Pod: Student's k8s cluster:

Dataset Description:

- Particulate Matter (PM) dataset for Delhi, India
- Area spanning 559 square kms.
- PM data recorded over three months from November 2020 to January 2021
- Data Source: https://www.cse.iitd.ac.in/pollutiondata/

Please go through the Lecture to know more details on this practice session.

1. Environment Setup

1. Create a project in the gitlab with name `lab11-data_intensive` under group `Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>`
2. Create k8s agent for this project, this is similar to the task performed in Lab 04

   Exercise 4 , Task 4.2

   a. This task should be performed in the Master (node1) node of the k8s cluster

   b. Give agent name as `lab11`

3. Install and register of Gitlab runners (build,deploy) as mentioned in Exercise 1 in Lab 07. Thi is to be done for the k8s-controller.
4. Setup NFS (Network File System) server in k8s-controller VM as mentioned below. This is required to store the dashboard and data source files required for the grafana

deployment on k8s cluster. This is a similar approach of using k8s *PersistentVolume* like in the previous labs.

a. Login to k8s-controller VM
b. Install using `sudo dnf install nfs-utils`
c. Start the service `sudo systemctl start nfs-server.service`
d. Enable the service `sudo systemctl enable nfs-server.service`
e. Check the status `sudo systemctl status nfs-server.service`
f. Now, make the directory `sudo mkdir -p /tmp/grafana-provisioning`
g. Now, enable the server to access the directory and files to client, `sudo vi /etc/exports` and paste the following line (Change the IP 172.17.89.20 to your k8s-master IP address) `/tmp/grafana-provisioning/ 172.17.89.20(rw,no_subtree_check,no_root_squash)`
h. Run the command `sudo exportfs -arv`
i. Change the ownership of the shared folder `sudo chmod 666 /tmp/grafana-provisioning`

5. Notes

Below some of the instructions are descriptive. That means you may get errors while executing the given minimal version of the commands. You need to investigate and fix those errors. For this, you may need to google and debug the error by yourself.

Remember that, we may see the commit history as well and the VMs while grading your submission.

2. Working with data intensive application- Job1

In this exercise, you are going to consume data from the already deployed data sources by us and perform simple data analytic activities such as preprocessing and processing to extract insights from the data.

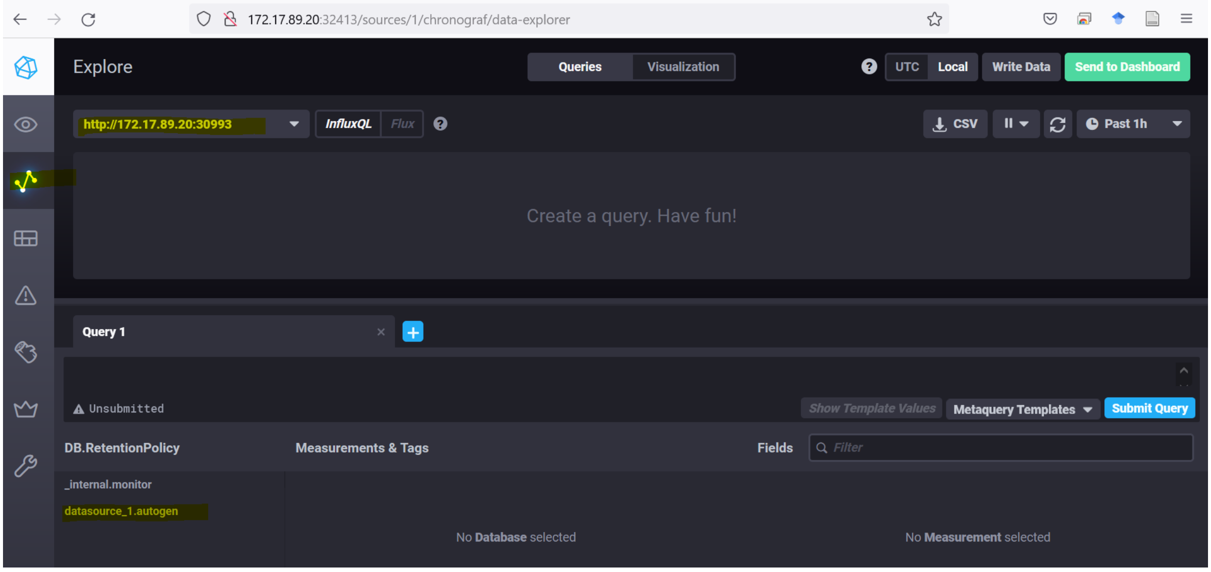
Step 2.1. Get acquainted with data source

The data used in these tasks are exported from the repository https://www.cse.iitd.ac.in/pollutiondata/publication and data consist of air quality with particle measurement values of size PM1_0, PM2_5,PM10. The PM values were measured from different locations in the city. The dataset consist of following features:

Feature Name	**Feature description**
uid	Unique Id for each record
dateTime	TimeStamp of the data recorded
deviceId	Sensor device ID
lat	Latitude - Location of the sensor
long	Longitude - Location of the sensor

pm1_0	Measured in micrograms per cubic meter
pm2_5	Measured in micrograms per cubic meter
pm10	Measured in micrograms per cubic meter

1. The data source is configured in VM with **DATA_SOURCE_IP: 172.17.89.20** and **PORT:**30993 using time series database influxdb.
2. We query the data using InfluxQL , It uses the structure as like
   TIMESTAMP MEASUREMENT TAG1, TAG2, FIELD1,FIELD2…
   Here, MEASUREMENT = AIR_QUALITY(Table name)
   TAG = (KEY=DEVICE_NAME, VALUE=16715D1)
   FIELD1=PM1.5
   FIELD2=PM2.0
3. Now you can query the data source and check for the list of recorded air quality data. For this task, you can use chronograph which provides visualisation to access the influxdb records and databases. Go to http://172.17.89.20:32413 and click on Explore, you will see the window as below and choose the data source as http://172.17.89.20:30993



172.17.89.20:32413/sources/1/chronograf/data-explorer

http://172.17.89.20:30993

Create a query. Have fun!

Unsubmitted

_internal.monitor

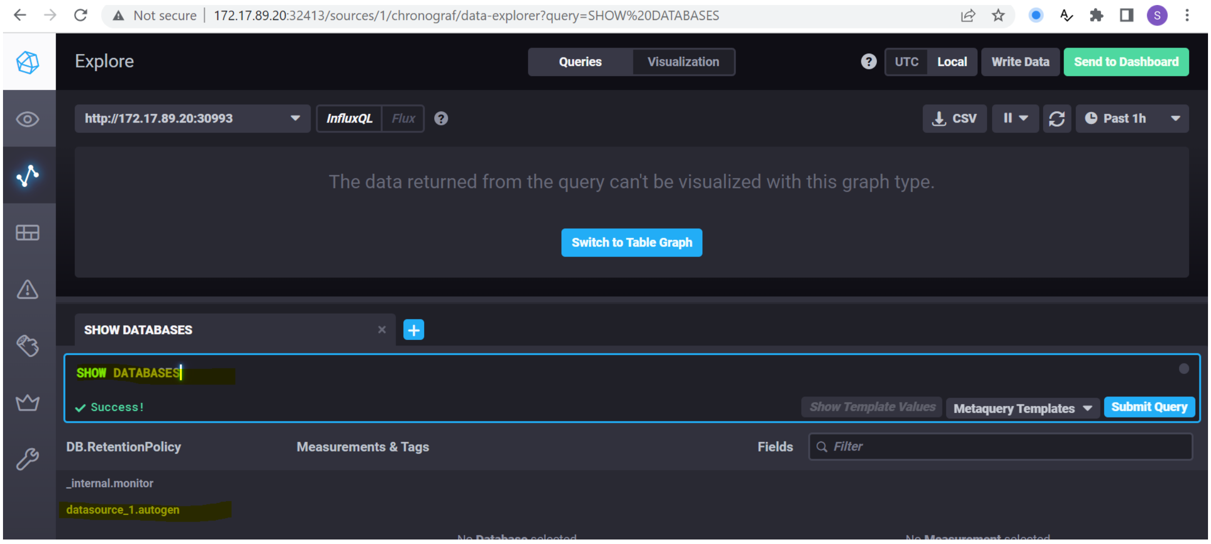
datasource_1.autogen

No Database selected

No Measurement selected

Now, let us write the queries to understand the data source.

a. Get the list of databases, Click on Meta_query Templates-->SHOW DATABASES
   You should see a list of databases as shown below, here **datasource_1** is the database of air quality monitoring systems.



Not secure | 172.17.89.20:32413/sources/1/chronograf/data-explorer?query=SHOW%20DATABASES

```
SHOW DATABASES
✓ Success!
```

```
_internal.monitor
datasource_1.autogen
```

b. Get the list of MEASUREMENTS from the database (Try Out yourself)
c. Get the TAGs from the database (Try Out yourself)
d. Query the data of specific TAG from the specific measurement.(Refer InfluxQL)

Step 2.2. Creating data processing tasks

Here, you're going to create a task to read air quality data from **datasource_1** and store local influxdb with grouping by the device id.
You can use Web IDE of gitlab to perform the following tasks

1. Create a directory as `job1` under the project's root directory.
2. Create a directory `data_preprocess` inside your `job1` directory and write a python script `job1/data_preprocess/data_preprocess.py` (sample code given below)
   In this task, you're going to read the data from data source **datasource_1** from **DATA_SOURCE_IP: 172.17.89.20:30993** and group the data according to device id and store it in your local influxdb in **k8s deployment**.

Filename:`/job1/data_preprocess/data_preprocess.py`

```
import pandas as pd
from influxdb import InfluxDBClient
from influxdb import DataFrameClient
import sys

ds_port = sys.argv[5]
ds_host = sys.argv[4]
ds_name = sys.argv[3]
local_influxdb_name = sys.argv[1]
local_influxdb_host = sys.argv[2]

# client to extract data from data_source
data_source = InfluxDBClient(host=ds_host, port=ds_port)

# Switch the database
```

```
data_source.switch_database(ds_name)
query = "SELECT * FROM air_quality;"

# Read data frame
df = pd.DataFrame(data_source.query(query).get_points())
df["time"] = pd.to_datetime(df["time"], errors="coerce")
# df = df.set_index('time')


data_sink = DataFrameClient(
       host=local_influxdb_host,
       port=8086
)

data_sink.drop_database(local_influxdb_name)
data_sink.create_database(local_influxdb_name)
data_sink.switch_database(local_influxdb_name)

for group, dataframe in df.groupby(["host"]):
       tags = {"host": group}
       fields = dataframe[["time", "pm1_0", "pm2_5", "pm10"]]
       fields = fields.set_index("time")
       data_sink.write_points(fields, 'preprocessed', tags, protocol="line")
```

3. Create a `requirements.txt` file to install necessary pip packages required to run the preprocessing service.

Filename:/job1/data_preprocess/requirements.txt

```
pandas
influxdb
```

4. Create a Dockerfile file `job1/data_preprocess/Dockerfile` (sample code given below) to build and deploy preprocess service containers.

Filename:/job1/data_preprocess/Dockerfile

```
FROM python:3.8-slim-buster
COPY requirements.txt requirements.txt


ARG INFLUX_DB_NAME
ARG INFLUX_DB_HOST
ARG DS_NAME
ARG DS_HOST
ARG DS_PORT


ENV LOCAL_INFLUX_DB=${INFLUX_DB_NAME}
ENV LOCAL_INFLUX_HOST=${INFLUX_DB_HOST}
ENV NAME=${DS_NAME}
ENV HOST=${DS_HOST}
ENV PORT=${DS_PORT}

RUN pip3 install -r requirements.txt
COPY . .
CMD ["/bin/bash", "-c","python3 data_preprocess.py $LOCAL_INFLUX_DB
$LOCAL_INFLUX_HOST $NAME $HOST $PORT"]
```

5. Create a `data_process` directory inside your project's root directory and write a python script `job1/data_process/data_process.py` (sample code given below) to process the data.

Filename: `/job1/data_process/data_process.py`

```
import pandas as pd
from influxdb import InfluxDBClient
from influxdb import DataFrameClient
import sys

local_influxdb_name = sys.argv[1]
local_influxdb_host = sys.argv[2]
print(local_influxdb_name,local_influxdb_host)
# client to extract data from data_source
data_sink = InfluxDBClient(
       host=local_influxdb_host,
       port=8086
)
data_sink.switch_database(local_influxdb_name)

query = "select * from preprocessed"
df = pd.DataFrame(data_sink.query(query).get_points())

df["time"] = pd.to_datetime(df["time"], errors="coerce")
df = df.set_index('time')

data_client = DataFrameClient(
       host=local_influxdb_host,
       port=8086
)
data_client.drop_database('processed')
data_client.create_database('processed')
data_client.switch_database('processed')

df_hourly =
df.groupby(['host']).resample('1H').mean().transform(pd.Series.interpolate).reset
_index()
```

```
for group, dataframe in df_hourly.groupby(["host"]):
        dataframe = dataframe.reset_index()
        tags = {"host": group}
        fields = dataframe[["time","pm1_0", "pm2_5", "pm10"]]
        fields = fields.set_index("time")
        data_client.write_points(fields, 'hourly', tags, protocol="line")


df_daily =
df.groupby(['host']).resample('D').mean().transform(pd.Series.interpolate).reset_
index()
print(df_daily)
for group, dataframe in df_daily.groupby(["host"]):
        dataframe = dataframe.reset_index()
        tags = {"host": group}
        fields = dataframe[["time","pm1_0", "pm2_5", "pm10"]]
        fields = fields.set_index("time")
        data_client.write_points(fields, 'daily', tags, protocol="line")

df_weekly =
df.groupby(['host']).resample('W').mean().transform(pd.Series.interpolate).reset_
index()
for group, dataframe in df_weekly.groupby(["host"]):
        dataframe = dataframe.reset_index()
        tags = {"host": group}
        fields = dataframe[["time","pm1_0", "pm2_5", "pm10"]]
        fields = fields.set_index("time")
        data_client.write_points(fields, 'weekly', tags, protocol="line")
```

6. Create a `job1/data_process/requirements.txt` file to install necessary pip packages required to run the data_processing service.

7. Create a Dockerfile file `job1/data_process/Dockerfile` (sample code given below) to build and deploy preprocess service pod.

Filename:`/job1/data_process/Dockerfile`

```
FROM python:3.8-slim-buster
COPY requirements.txt requirements.txt

ARG INFLUX_DB_NAME
ARG INFLUX_DB_HOST

ENV INFLUX_DB=${INFLUX_DB_NAME}
ENV INFLUX_HOST=${INFLUX_DB_HOST}
RUN pip3 install -r requirements.txt
```

```
COPY . .
CMD ["/bin/bash", "-c","python3 data_process.py $INFLUX_DB
$INFLUX_HOST"]
```

8. Now let us create a `.gitlab-ci.yml` (with below sample code) CI file to build the above services and run the job1 tasks. Change the values according to your deployment.
   You have to build two images for job1, i.e data_preprocess, and data_process.
   Further, you can tag the images as job1_data_preprocess and job1_data_process

Filename: `.gitlab-ci.yml`

```
variables:
  DS_HOST: "172.17.89.20"
  DS_PORT: "30993"
  DS_NAME: datasource_1
  IMAGE_HUB: "gitlab.cs.ut.ee:5050/poojara/"
  IMAGE_JOB1_PREPROCESS: "gitlab.cs.ut.ee:5050/devops2022-fall/all-solutions/lab11-data_intensive/job1_data_preprocess"
  IMAGE_JOB1_PROCESS: "gitlab.cs.ut.ee:5050/devops2022-fall/all-solutions/lab11-data_intensive/job1_data_process"
  INFLUX_DB_HOST: "influxdb-service"
  INFLUX_DB_NAME: "local"
  INFLUX_DB_PORT: "8086"


stages:
  - build
  - deploy


job1_build:
  script:
     # 1. Building the docker image for data_preprocess
    - docker login -u poojara -p $gitlabpassword  $IMAGE_HUB
    - docker build -t $IMAGE_JOB1_PREPROCESS:latest -t $IMAGE_JOB1_PREPROCESS:$CI_COMMIT_SHORT_SHA --build-arg  DS_HOST=$DS_HOST
--build-arg  DS_PORT=$DS_PORT --build-arg  DS_NAME=$DS_NAME --build-arg  INFLUX_DB_HOST=$INFLUX_DB_HOST  --build-arg
INFLUX_DB_NAME=$INFLUX_DB_NAME  -f ./job1/data_preprocess/Dockerfile ./job1/data_preprocess
    - docker push $IMAGE_JOB1_PREPROCESS
     # 2. Building the docker image for data_process (This commands  to be written by you as like previous)

stage: build
  tags:
    - build
```

9. We will deploy the following services to the k8s cluster. The sample deployment file can be downloaded from here. Download, modify if required and add to the project's root directory `/services_deployment.yml`
   In this file, update the *server: 172.17.89.20* (line # 80) to your K8s-controller external IP address. This is used as the address of the nfs server.
   Following are the overview of the services used in this experiments.

Service Name	Port	Usage
InfluxDB	8086	Store the processed air quality data

Chronograf	8888	Visualization service for Influxdb
Grafana	3000	Visualize the insights from the data.

Further, update the `.gitlab-ci.yml` file to deploy the services to k8s-cluster. This is similar to the previous CI file used in Lab 07. Please update the kubectl config and secrets commands with associated variable values.

Filename: `.gitlab-ci.yml`

```
services_deploy:
    stage: deploy
    image:
        name: bitnami/kubectl:latest
        entrypoint: [""]
    script:
        - kubectl config use-context devops2022-fall/all-solutions/lab11-data_intensive:lab11
        - kubectl delete secrets registry-credentials || true
        - kubectl create secret docker-registry registry-credentials --docker-server=https://gitlab.cs.ut.ee:5050
--docker-username=poojara --docker-password=$gitlabpassword --docker-email=poojara@ut.ee || true
        - kubectl delete -f ./services_deployment.yml || true
        - kubectl apply -f ./services_deployment.yml
    tags:
        - deploy
```

10. Further, create k8s deployment files to deploy the data_preprocess and data_process tasks of job1. The images are built and pushed in step 9. Here, change the names in the `image` field and others, if any.

Filename:`/job_deployment.yml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: preprocess-deployment
  labels:
    app: data-preprocess-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: data-preprocess-app
  template:
    metadata:
      labels:
        app: data-preprocess-app
    spec:
      containers:
      - name: preprocessapp
        image: <job1_preprocess_image>
---
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: process-deployment
  labels:
    app: data-process-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: data-process-app
  template:
    metadata:
      labels:
        app: data-process-app
    spec:
      containers:
      - name: processapp
        image: <job1_process_image>
```

11. Update the `.gitlab-ci.yml` file to deploy the job1 tasks.

Filename: `/.gitlab-ci.yml`

```
job1_deploy:
    stage: deploy
    image:
        name: bitnami/kubectl:latest
        entrypoint: [""]
    script:
        - kubectl config use-context devops2022-fall/all-solutions/lab11-data_intensive:lab11
        - kubectl delete -f ./job_deployment.yml || true
        - kubectl apply -f ./job_deployment.yml

    tags:
        - deploy
```

12. Commit the code with message “job1 data processing tasks added”.
13. Now, let us create a datasource and data dashboard templates directory for grafana service in nfs storage.
    Login to k8s-controller VM.
    a. Create sub directories as *datasources* in `/tmp/grafana-provisioning/`*datasources* and *dashboards* as `/tmp/grafana-provisioning/`*dashboards*
    b. Create a file to add a data source

Filename: `/tmp/grafana-provisioning/datasources/datasource.yml`

```
apiVersion: 1
datasources:
  - name: job1
    type: influxdb
    access: proxy
    database: local
    url: http://influxdb-service:8086
    isDefault: true
    editable: true
```

c. Create a file for grafana dashboard template

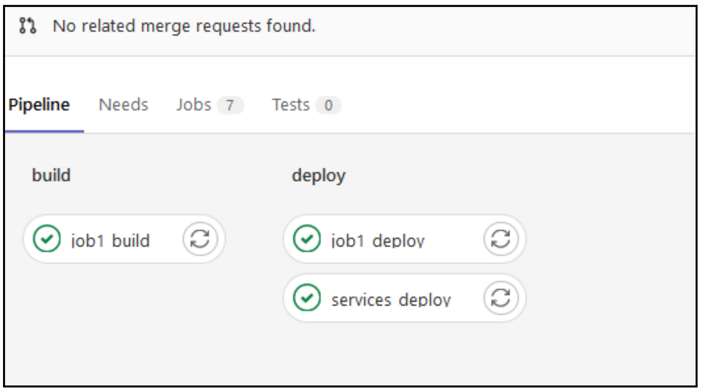
Filename: /tmp/grafana-provisioning/dashboards/dashboard.yml

```
apiVersion: 1
providers:
- name: InfluxDB
  folder: ''
  type: file
  disableDeletion: false
  editable: true
  options:
    path: /etc/grafana/provisioning/dashboards
```

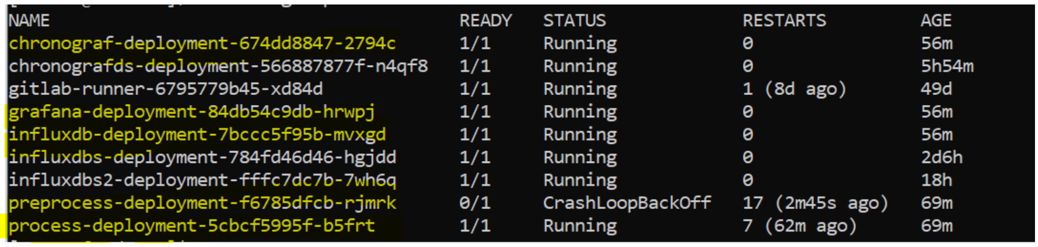
d. Create three files that are basically dashboard templates of grafana and need to stored in the location /tmp/grafana-provisioning/dashboards/ and ypu can refer the data from https://gitlab.cs.ut.ee/devops22fallpub/lab11-data-intensive in the following files. (Good Idea is to clone repo and use the files in the required location)

   i. hourly.json
   ii. weekly.json
   iii. daily.json

14. Run the pipeline.
15. Check for the completion of pipeline execution.

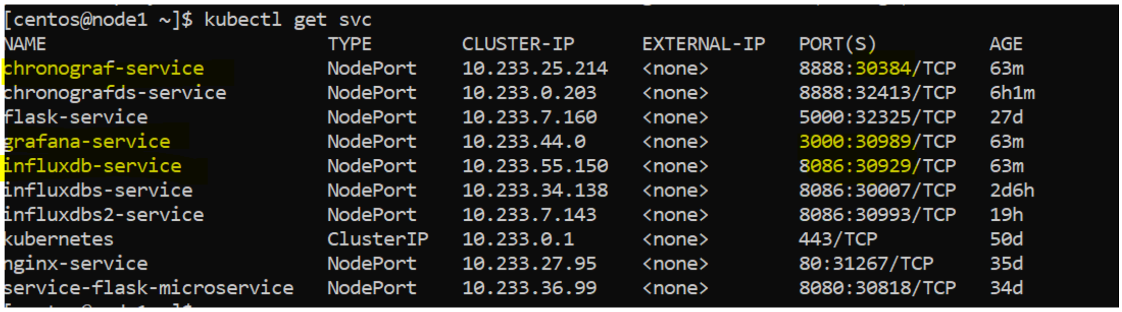


16. Now, login to k8s-Master VM and check the running pods



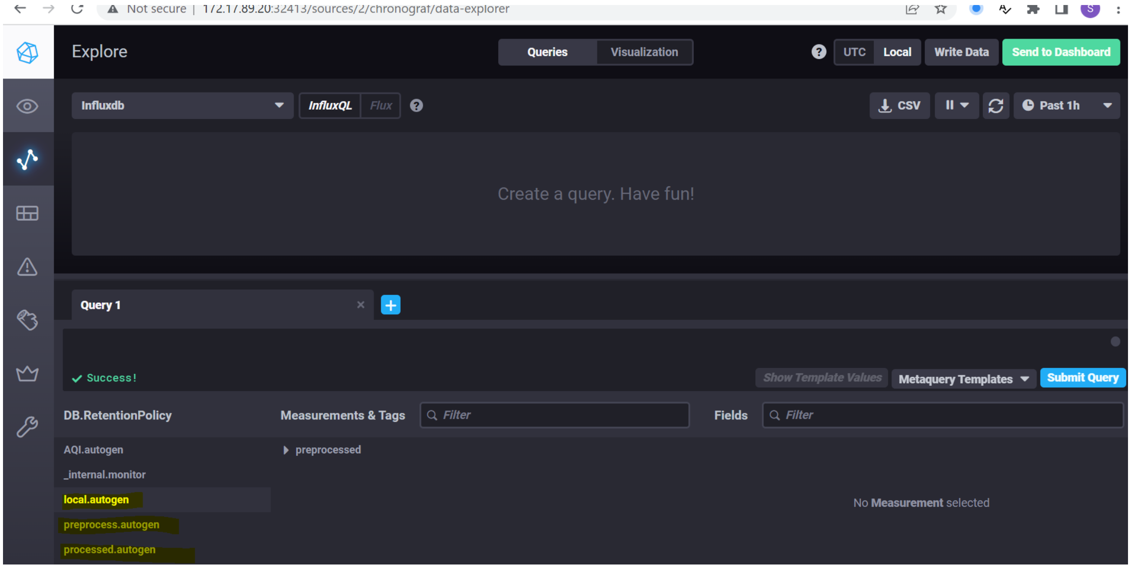
```
NAME                                    READY   STATUS             RESTARTS        AGE
chronograf-deployment-674dd8847-2794c   1/1     Running            0               56m
chronografds-deployment-566887877f-n4qf8 1/1    Running            0               5h54m
gitlab-runner-6795779b45-xd84d          1/1     Running            1 (8d ago)      49d
grafana-deployment-84db54c9db-hrwpj     1/1     Running            0               56m
influxdb-deployment-7bccc5f95b-mvxgd    1/1     Running            0               56m
influxdbs-deployment-784fd46d46-hgjdd   1/1     Running            0               2d6h
influxdbs2-deployment-fffc7dc7b-7wh6q   1/1     Running            0               18h
preprocess-deployment-f6785dfcb-rjmrk   0/1     CrashLoopBackOff   17 (2m45s ago)  69m
process-deployment-5cbcf5995f-b5frt     1/1     Running            7 (62m ago)     69m
```

17. Check the node port address for your services `kubectl get svc`



```
[centos@node1 ~]$ kubectl get svc
NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
chronograf-service            NodePort    10.233.25.214   <none>        8888:30384/TCP   63m
chronografds-service          NodePort    10.233.0.203    <none>        8888:32413/TCP   6h1m
flask-service                 NodePort    10.233.7.160    <none>        5000:32325/TCP   27d
grafana-service               NodePort    10.233.44.0     <none>        3000:30989/TCP   63m
influxdb-service              NodePort    10.233.55.150   <none>        8086:30929/TCP   63m
influxdbs-service             NodePort    10.233.34.138   <none>        8086:30007/TCP   2d6h
influxdbs2-service            NodePort    10.233.7.143    <none>        8086:30993/TCP   19h
kubernetes                    ClusterIP   10.233.0.1      <none>        443/TCP          50d
nginx-service                 NodePort    10.233.27.95    <none>        80:31267/TCP     35d
service-flask-microservice    NodePort    10.233.36.99    <none>        8080:30818/TCP   34d
```

18. Open your chronograf service http://k8s_MASTER_VM_EXT_IP:NODE_PORT_ADDRESS_CHRONOGRAF and you should see the preprocessed and processed data as shown below

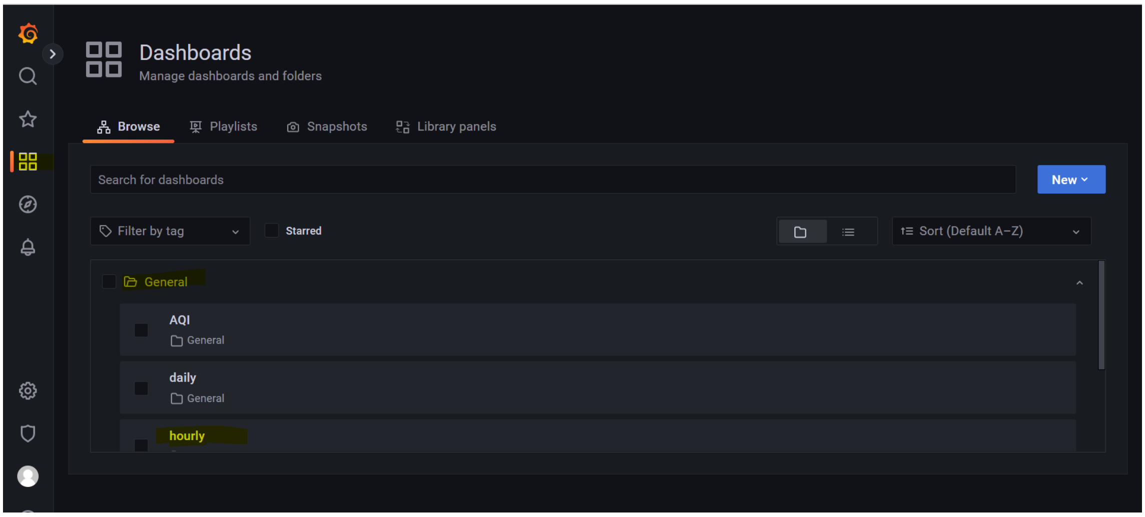


Not secure | 172.17.89.20:32413/sources/2/chronograf/data-explorer

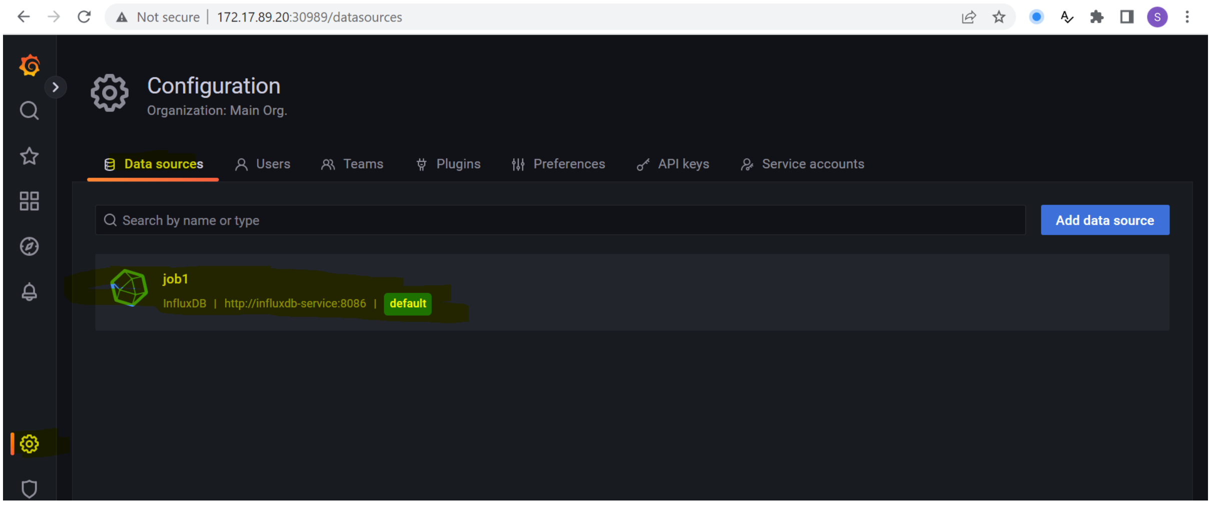
19. Check in **k8s-master VM** for the status of the services running or errors.
    a. **job1-preprocess-deployment** and **job1-process-deployment** pods will exit into CrashLoopBackOff after completing the job and need not to worry about.
    b. Check for the logs of **job1-preprocess-deployment** and **job1-process-deployment** pods for any error messages in your python programs.

Step 2.3. Data visualization in grafana

After successful deployment of all services, you should see the grafana service running in the **k8s cluster**. Now, goto grafana service and open the dashboards. Access the grafana service http://k8s_MASTER_VM_EXT_IP:NODE_PORT_ADDRESS_GRAFANA
Grafana uses default **username:admin** and **password:admin.**
You can access the dashboards as shown below:

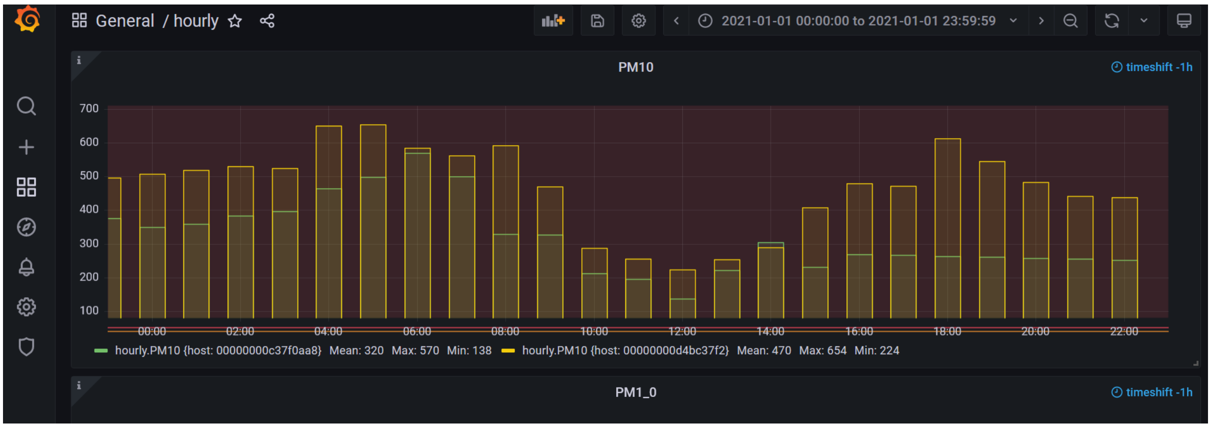


You can check the data source as shown below:



The following are the processed data visualized using the dashboards:

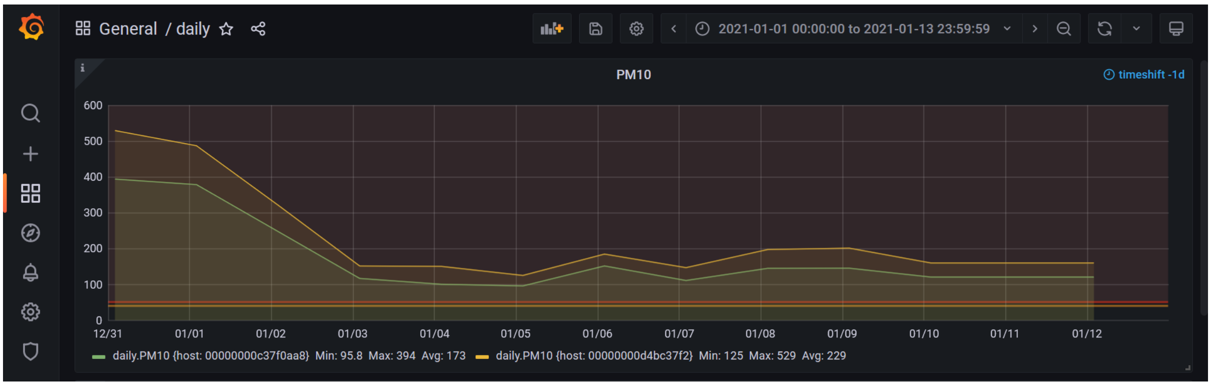
Hourly dashboard(hourly.json):


General / hourly
2021-01-01 00:00:00 to 2021-01-01 23:59:59
PM10
timeshift -1h
700
600
500
400
300
200
100
00:00
02:00
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
hourly.PM10 {host: 00000000c37f0aa8} Mean: 320 Max: 570 Min: 138
hourly.PM10 {host: 00000000d4bc37f2} Mean: 470 Max: 654 Min: 224
PM1_0
timeshift -1h

Screenshot - 1

Take a screenshot of a webpage and the IP address should be clearly visible.

Daily dashboard(daily.json):



Screenshot - 2

Take a screenshot of a webpage and the IP address should be clearly visible.

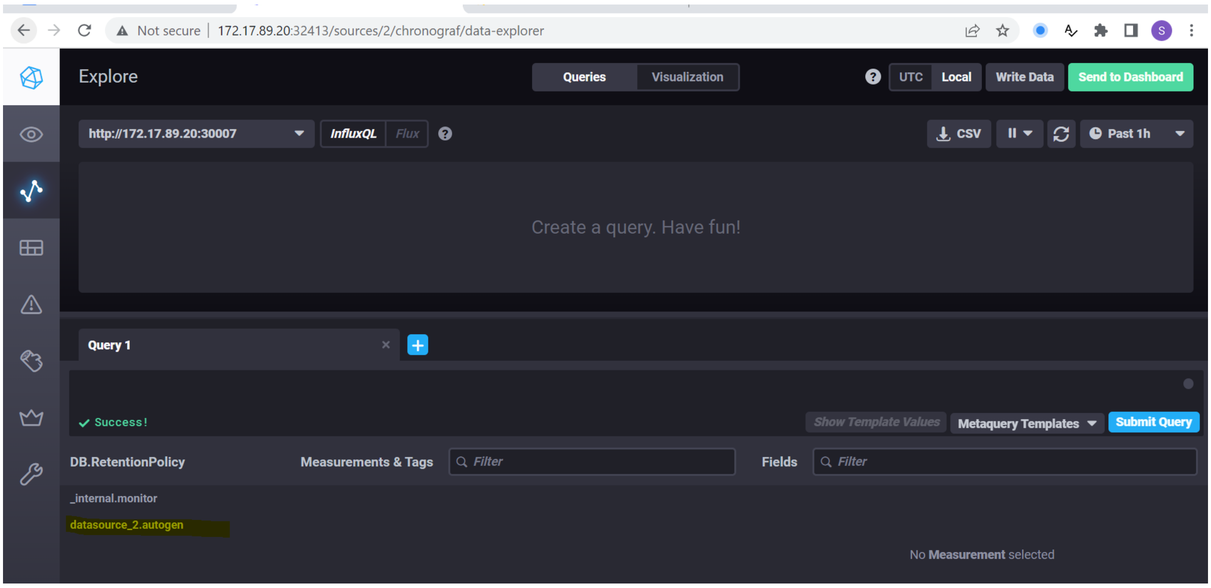
3. Working with data intensive application- Job2

In this exercise, you are going to consume data from the already deployed data sources datasource_2 and calculate Air Quality Index(AQI). AQI is calculated based on the number of occurrences of PM 2.5 and PM 10.

Step 3.1. Get acquainted with datasource_2

We will work with another data source similar to data used in Exercise 2.

1. Now you can query the data source and check for the list of recorded air quality data. For this task, you can use chronograph which provides visualization to access the influxdb records and databases. Go to http://172.17.89.20:31734 and you will see the window as below and choose the data source as http://172.17.89.20:30007


Not secure | 172.17.89.20:32413/sources/2/chronograf/data-explorer

http://172.17.89.20:30007

Create a query. Have fun!

✓ Success!

DB.RetentionPolicy

_internal.monitor

datasource_2.autogen

No Measurement selected

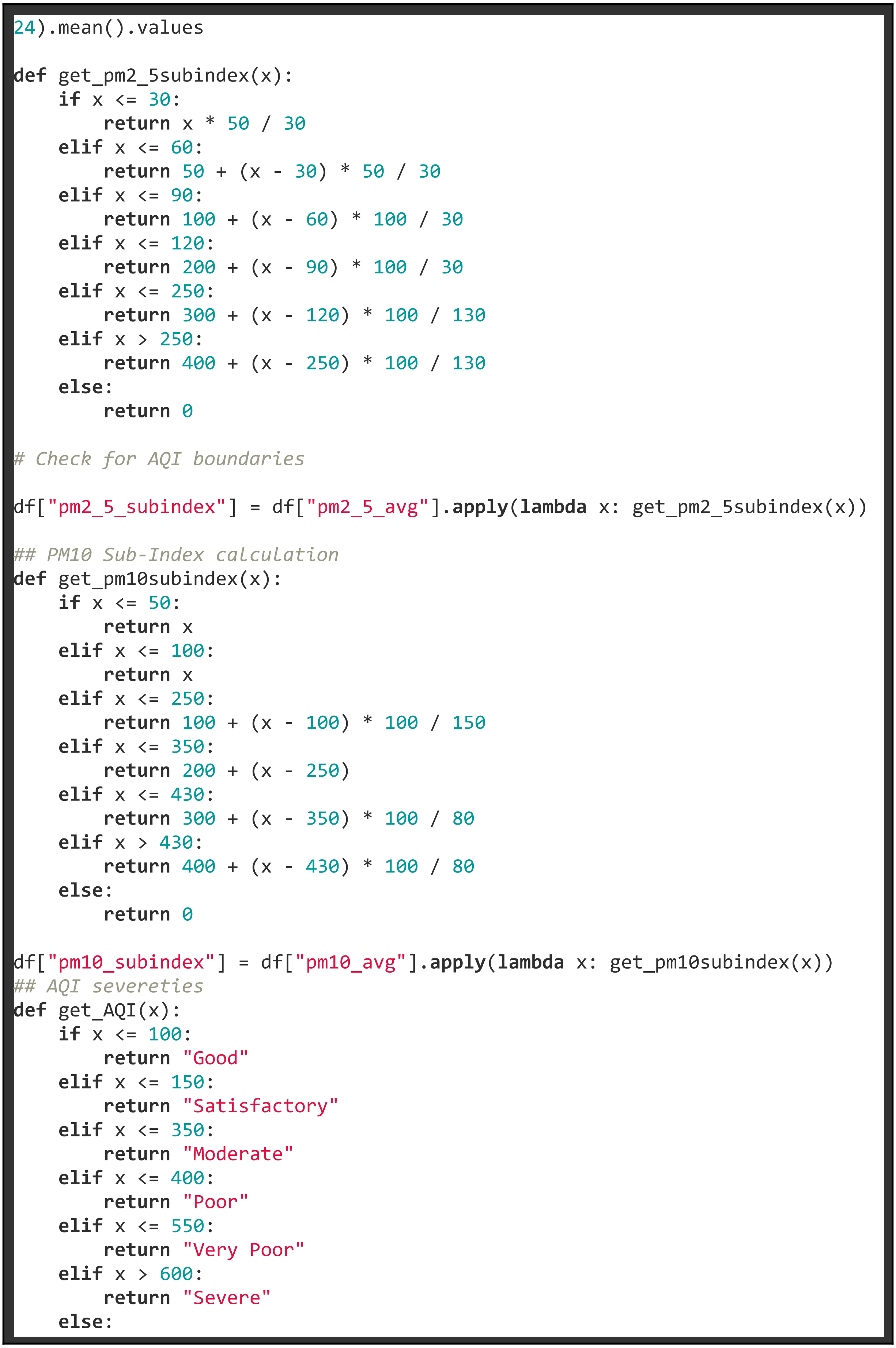
Step 3.2. Creating data analytic job2

Here, you're going to create a task to read air quality data from **datasource_2** and store local influxdb with air quality index. Here we use similar approach as used in Exercise 2. You can use gitlab Web IDE to perform the tasks.

1. Create a directory for as job2 under project's root directory `Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>/lab11-data_intensive/job2`
2. Create a directory `data_preprocess` inside your job2 directory and copy all the content from `job1/data_preprocess` directory.
3. Create a directory `data_process` inside your job2 directory and copy all the content from `job1/data_process` as it is similar.
   But we need to modify the `data_process.py` to calculate AQI as below.

Filename: `Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>/lab11-data_intensive/job2/data_process.py`

```
# Read sampled data
# Classify - Average of hourly, daily and monthly data
# Insert the classfied data in influxdb

import pandas as pd
from influxdb import InfluxDBClient
from influxdb import DataFrameClient
import sys
import numpy as np

# Get the variables
local_influxdb_name = sys.argv[1]
local_influxdb_host = sys.argv[2]
print(local_influxdb_name,local_influxdb_host)

# client to extract data from data_source
data_sink = InfluxDBClient(
    host=local_influxdb_host,
    port=8086
)
data_sink.switch_database(local_influxdb_name)

# get preprocesed data
query = "select * from preprocessed"
df = pd.DataFrame(data_sink.query(query).get_points())

df["time"] = pd.to_datetime(df["time"], errors="coerce")
df = df.set_index('time')


# Calculating AQI
# 1. Take rolling mean of 24h
df["pm10_avg"] = df.groupby("host")["pm10"].rolling(window =
24).mean().values

df["pm2_5_avg"] = df.groupby("host")["pm2_5"].rolling(window =
```

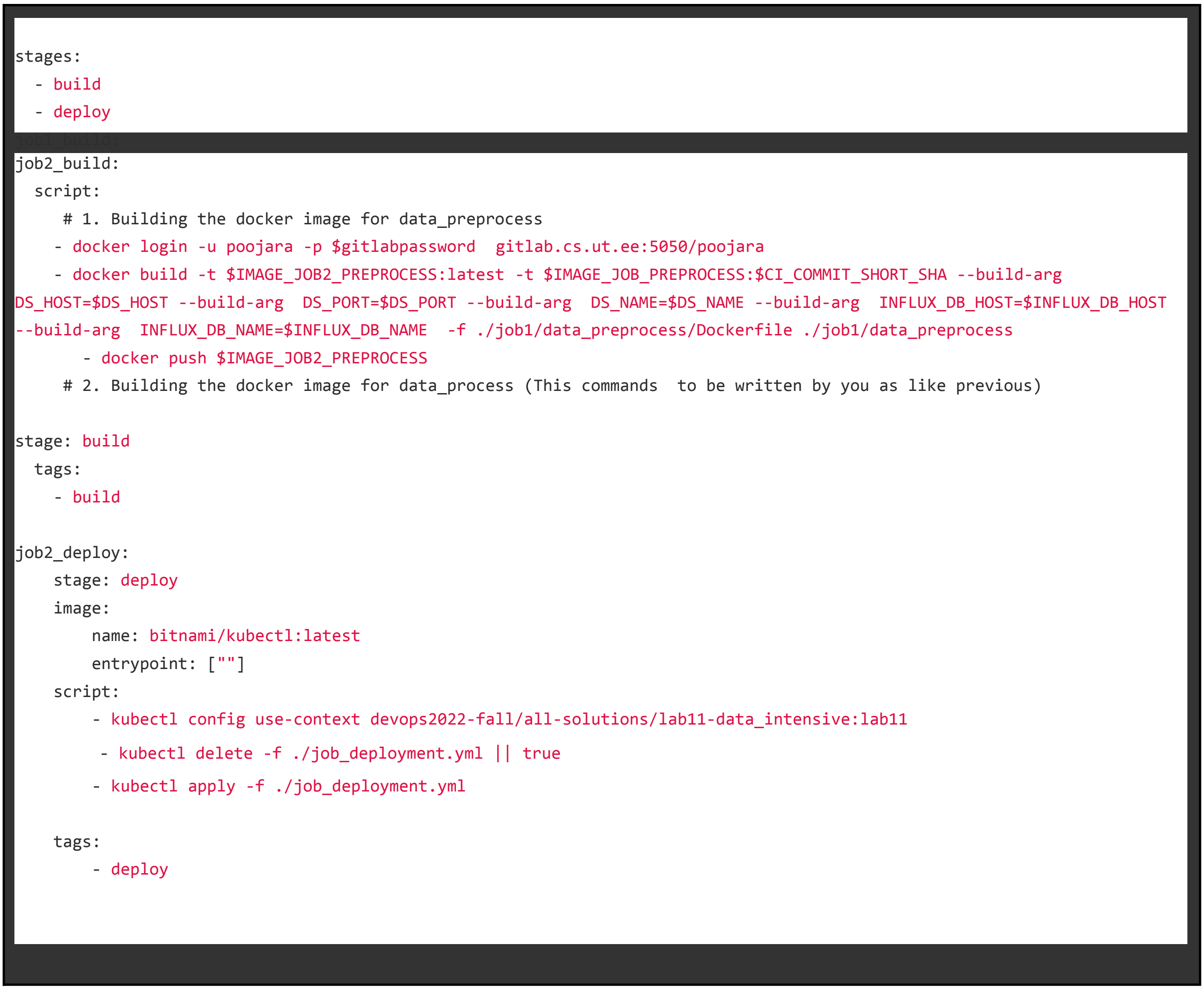


```python
24).mean().values

def get_pm2_5subindex(x):
    if x <= 30:
        return x * 50 / 30
    elif x <= 60:
        return 50 + (x - 30) * 50 / 30
    elif x <= 90:
        return 100 + (x - 60) * 100 / 30
    elif x <= 120:
        return 200 + (x - 90) * 100 / 30
    elif x <= 250:
        return 300 + (x - 120) * 100 / 130
    elif x > 250:
        return 400 + (x - 250) * 100 / 130
    else:
        return 0

# Check for AQI boundaries

df["pm2_5_subindex"] = df["pm2_5_avg"].apply(lambda x: get_pm2_5subindex(x))

## PM10 Sub-Index calculation
def get_pm10subindex(x):
    if x <= 50:
        return x
    elif x <= 100:
        return x
    elif x <= 250:
        return 100 + (x - 100) * 100 / 150
    elif x <= 350:
        return 200 + (x - 250)
    elif x <= 430:
        return 300 + (x - 350) * 100 / 80
    elif x > 430:
        return 400 + (x - 430) * 100 / 80
    else:
        return 0

df["pm10_subindex"] = df["pm10_avg"].apply(lambda x: get_pm10subindex(x))
## AQI severeties
def get_AQI(x):
    if x <= 100:
        return "Good"
    elif x <= 150:
        return "Satisfactory"
    elif x <= 350:
        return "Moderate"
    elif x <= 400:
        return "Poor"
    elif x <= 550:
        return "Very Poor"
    elif x > 600:
        return "Severe"
    else:
        return np.NaN


df["Checks"] = (df["pm2_5_subindex"] > 0).astype(int) + \
                (df["pm10_subindex"] > 0).astype(int)


df["AQI"] = round(df[["pm2_5_subindex", "pm10_subindex"]].max(axis = 1))
df.loc[df["pm2_5_subindex"] + df["pm10_subindex"] <= 0, "AQI"] = np.NaN
df.loc[df.Checks <= 1, "AQI"] = np.NaN

df["AQI_calculated"] = df["AQI"].apply(lambda x: get_AQI(x))

print(df[~df.AQI.isna()].head(13))

data_client = DataFrameClient(
    host=local_influxdb_host,
    port=8086
)

data_client.drop_database('AQI')
data_client.create_database('AQI')
data_client.switch_database('AQI')


for group, dataframe in df.groupby(["host"]):
    dataframe = dataframe.reset_index()
    tags = {"host": group}
    fields = dataframe[["time","pm1_0", "pm2_5",
"pm10","AQI","AQI_calculated"]]
    fields = fields.set_index("time")
    data_client.write_points(fields, 'AQI', tags, protocol="line")
```

4. Now let us update a `.gitlab-ci.yml` file to build an image for job2.

Filename: `Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>/lab11-data_intensive/.gitlab-ci.yml`

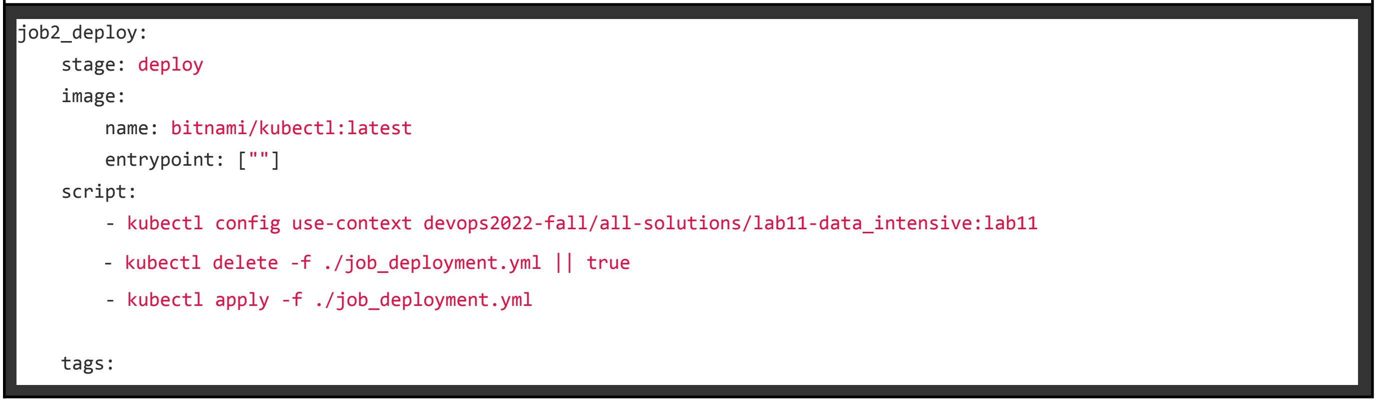
```
variables:
  DS_HOST: "172.17.89.20"
  DS_PORT: "30007"
  DS_NAME: datasource_2
  IMAGE_HUB: "gitlab.cs.ut.ee:5050/poojara/"
  IMAGE_JOB2_PREPROCESS: "gitlab.cs.ut.ee:5050/devops2022-fall/all-solutions/lab11-data_intensive/job2_data_preprocess"
  IMAGE_JOB2_PROCESS: "gitlab.cs.ut.ee:5050/devops2022-fall/all-solutions/lab11-data_intensive/job2_data_process"
  INFLUX_DB_HOST: "influxdb-service"
  INFLUX_DB_NAME: "preprocess"
  INFLUX_DB_PORT: "8086"
```



```
stages:
  - build
  - deploy
job2_build:
  script:
     # 1. Building the docker image for data_preprocess
    - docker login -u poojara -p $gitlabpassword  gitlab.cs.ut.ee:5050/poojara
    - docker build -t $IMAGE_JOB2_PREPROCESS:latest -t $IMAGE_JOB_PREPROCESS:$CI_COMMIT_SHORT_SHA --build-arg
DS_HOST=$DS_HOST --build-arg  DS_PORT=$DS_PORT --build-arg  DS_NAME=$DS_NAME --build-arg  INFLUX_DB_HOST=$INFLUX_DB_HOST
--build-arg  INFLUX_DB_NAME=$INFLUX_DB_NAME  -f ./job1/data_preprocess/Dockerfile ./job1/data_preprocess
       - docker push $IMAGE_JOB2_PREPROCESS
     # 2. Building the docker image for data_process (This commands  to be written by you as like previous)

stage: build
  tags:
    - build

job2_deploy:
    stage: deploy
    image:
        name: bitnami/kubectl:latest
        entrypoint: [""]
    script:
        - kubectl config use-context devops2022-fall/all-solutions/lab11-data_intensive:lab11

         - kubectl delete -f ./job_deployment.yml || true

        - kubectl apply -f ./job_deployment.yml


    tags:
        - deploy
```

5. Update *`image`* field in the `job_deployment .yml` with corresponding `job2_data_preprocess` and `job2_data_process`
6. Now let us update a `.gitlab-ci.yml` file to deploy an image for job2.

Filename: Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>/lab11-data_intensive/.gitlab-ci.yml



```
job2_deploy:
    stage: deploy
    image:
        name: bitnami/kubectl:latest
        entrypoint: [""]
    script:
        - kubectl config use-context devops2022-fall/all-solutions/lab11-data_intensive:lab11

        - kubectl delete -f ./job_deployment.yml || true

        - kubectl apply -f ./job_deployment.yml


    tags:
      - deploy
```

7. Commit the code with message “job2 data processing tasks added”
8. Login to k8s-controller VM and
   a. Create a file to add a data source

Filename: `/tmp/grafana-provisioning/datasources/datasource.yml`

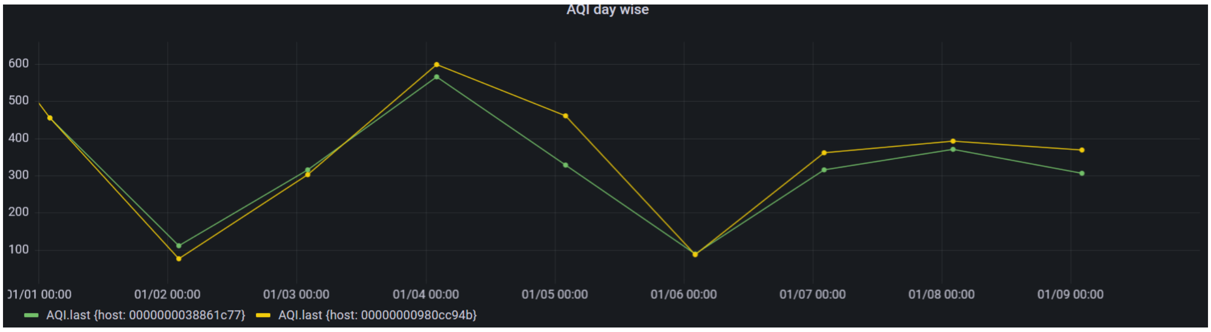
```
apiVersion: 1
datasources:
  - name: job1
    type: influxdb
    access: proxy
    database: local
    url: http://influxdb-service:8086
    isDefault: false
    editable: true

  - name: job2
    type: influxdb
    access: proxy
    Database: AQI
    url: http://influxdb-service:8086
    isDefault: true
    editable: true
```

   b. Create a `AQI.json` that is basically dashboard template of grafana and need to stored in the location `/tmp/grafana-provisioning/dashboards/AQI.json` and you can use the dashboard template from here.
9. Now run the pipeline again.

Step 3.3. Data visualization in grafana

After successful deployment of all services, you should see the grafana service running with AQI hourly, AQI day wise dashboards. If you're unable to see the dashboards properly then please check if the influxdb datasource is configured properly or not.

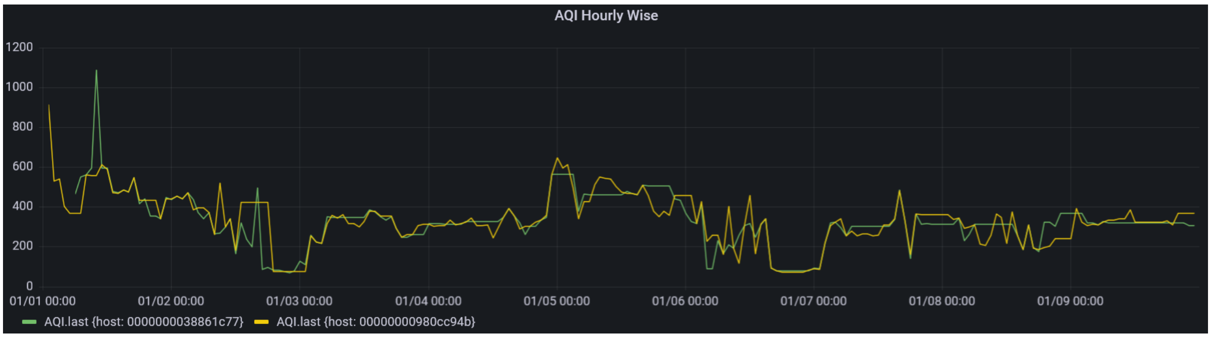
AQI day wise:



Screenshot - 3

Take a screenshot of a webpage and the IP address should be clearly visible.

AQI hourly wise of single day:



Screenshot - 4

Take a screenshot of a webpage and the IP address should be clearly visible.

4. Homework: Working with data-intensive application- Job3

In this exercise, you are going to set up your own data source with the name **datasource_3** and try to process the data by making a cluster into four groups as shown below and try to plot the bar graph in grafana with these clusters of a particular day like 'Night', 'Morning', 'Afternoon', 'Evening.'

Step 4.1. Setting up of data source *datasource_3*

1. Download the dataset to **k8s-controller VM** from the repository https://owncloud.ut.ee/owncloud/index.php/s/sgMEd5XS3g3RrWT
2. Create a docker influxdb container in **k8s-controller VM** using the following command
   ```
   sudo docker run -d  -p 8087:8086  --name influxdb -v influxdb:/var/lib/influxdb2 influxdb:1.1.1
   ```
3. Unzip the dataset and run the following `csvToInfluxdb.py` code to insert data into influxdb.
   Note:
   1. Install pandas and influxdb using pip in k8s-controller VM if required.
   2. This task can also be done in your local machine but needs to change the IP address of the host.

Filename: `csvToInfluxdb.py`

```
import pandas as pd
from influxdb import InfluxDBClient
import os
import glob
import sys

#dbname = sys.argv[1]

dbname = "datasource_3"
client = InfluxDBClient(host=localhost, port=8087)

# Check for existing database
client.drop_database(dbname)
client.create_database(dbname)
client.switch_database(dbname)

#filename = sys.argv[2]
path = os.getcwd()
csv_files = glob.glob(os.path.join(path, "*.csv"))

print(csv_files)
for f in csv_files:

#       file_path = open(filename,'r')
        csvReader = pd.read_csv(f)
        data = []
        for row_index, row in csvReader.iterrows() :
                tags = row[3]
                json_body = {
                "measurement": "air_quality",
                "time": row[2],
                "tags": {
                        "host": tags
                        },
                "fields": {
                        "lat":row[4],
                        "long": row[5],
                        "pm1_0": row[6],
                        "pm2_5": row[7],
                        "pm10" : row[8]
                                }
                        }
                data.append(json_body)
        client.write_points(data)
print("Data inserted successfully")
```

4. Test your inserted data in influxdb.
    a. Login to K8s Master VM and run the command `kubectl get svc and` note down, node port address of chronograf service

b. Open Chronograf service http://<k8s_Master_EXT_IP>:NODE_PORT_ADDRESS

c. Add data source as http://localhost:8087

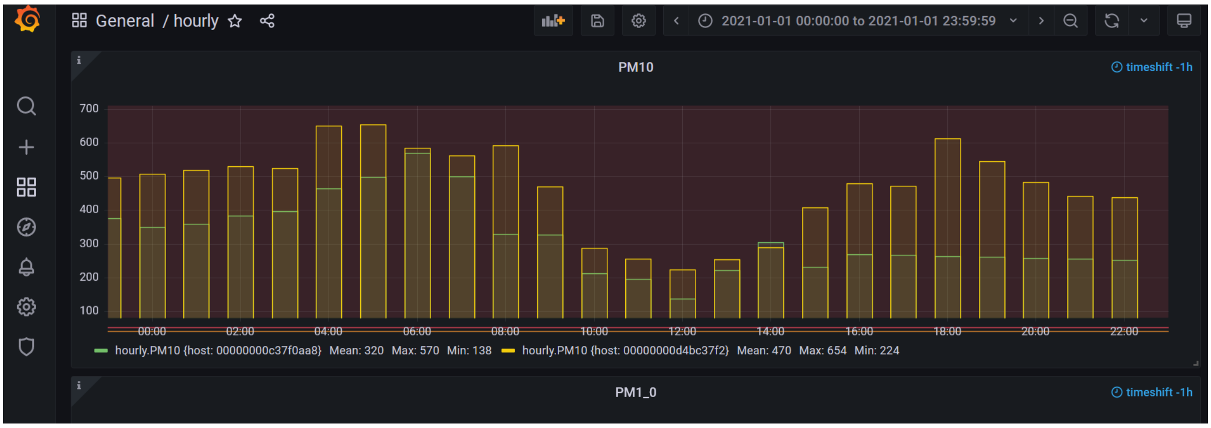
d. Click on Explore from the left panel.

5. Now you can query the data source and check for the list of recorded air quality data. For this task, you can use chronograph which provides visualization to access the influxdb records and databases. You need to add a datasource as http://localhost:8087

Step 4.2. Setting up of job3

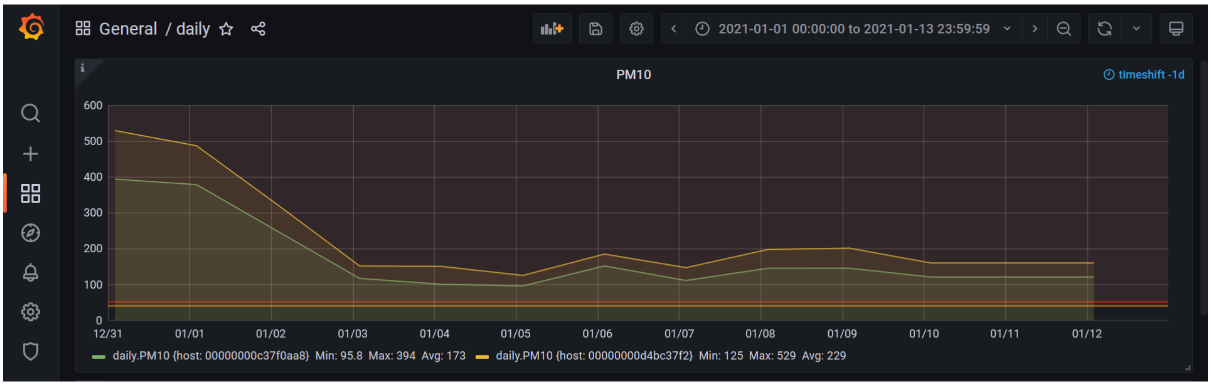
Here, you're going to create a task to read air quality data from **datasource_3** and stored in local influxdb(**deployment VM**) (This task is very similar to Exercise 2).

1. Create a directory `job3` and copy the contents of the directory of job1 for preprocessing and performing data processing activities.
2. In the `.gitlab-ci.yml` file update to run job3 (DS_NAME:datasource_3)
3. Now, commit with a message as “job3-1 data processing tasks added”
4. Now, goto grafana service and open the dashboards to visualise the hourly, daily and weekly data.


General / hourly
2021-01-01 00:00:00 to 2021-01-01 23:59:59
PM10
timeshift -1h
700
600
500
400
300
200
100
00:00
02:00
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
hourly.PM10 {host: 00000000c37f0aa8} Mean: 320 Max: 570 Min: 138
hourly.PM10 {host: 00000000d4bc37f2} Mean: 470 Max: 654 Min: 224
PM1_0
timeshift -1h

Screenshot - 5

Take a screenshot of a webpage and the IP address should be clearly visible.


General / daily
2021-01-01 00:00:00 to 2021-01-13 23:59:59
PM10
timeshift -1d
600
500
400
300
200
100
0
12/31
01/01
01/02
01/03
01/04
01/05
01/06
01/07
01/08
01/09
01/10
01/11
01/12
daily.PM10 {host: 00000000c37f0aa8} Min: 95.8 Max: 394 Avg: 173
daily.PM10 {host: 00000000d4bc37f2} Min: 125 Max: 529 Avg: 229

Screenshot - 6

`Take a screenshot of a webpage and the IP address should be clearly visible.`

Step 4.3. Update the job3

1. Update `data_process.py` to aggregate the data according to clusters and store the values in the influxdb and access in grafana. You can use similar logic as below:

Filename: `Devops2022fall/students/devops2022Fall-<lastname>-<studyCode>/lab11-data_intensive/job3/data_process/data_process.py`

```
import time
# 1. Read data from influxdb running deployment VM
# 2. Make the clusters
session=pd.cut(df.time.dt.hour,
             [0,6,12,18,23],
             labels=["Night","Morning","Afternoon","Evening"],
             include_lowest=True)

df['time_interval'] =session
df['time_interval'] = df['time_interval'].astype(str)

#3. Insert the data frame to Influxdb running in deployment VM
dbname = cluster
# Code for inserting
for group,dataframe in df.groupby(["time_interval"]):
      for host, df_ in dataframe.groupby(["host"]):
      df_ = df_.reset_index()
      tags = {"time_interval": group,"host":host}
      fields = df_[["time","pm1_0", "pm2_5", "pm10"]]
      fields = fields.set_index("time")
      client.write_points(fields, 'cluster', tags, protocol="line")
      time.sleep(1)
```

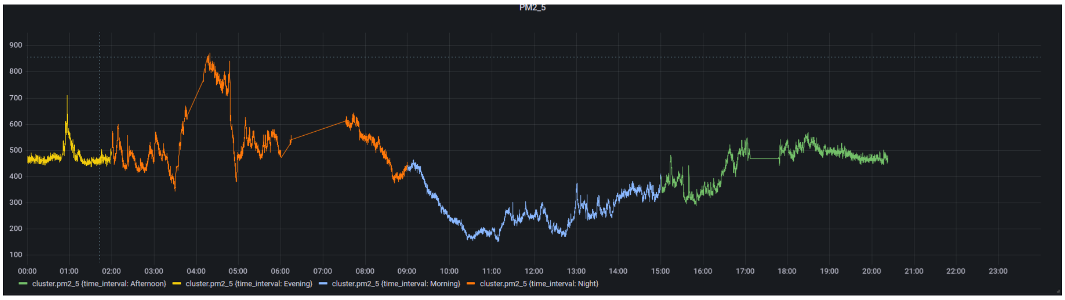
2. Login to k8s-contoller VM and
   a. Modify the `grafana-provisioning/datasources/datasource.yml`

Filename: `grafana-provisioning/datasources/datasource.yml`

```
apiVersion: 1
datasources:
  - name: job1
      type: influxdb
      access: proxy
      database: poojara
      url: http://influxdb-service:8086
      isDefault: false
      editable: true

  - name: job2
      type: influxdb
      access: proxy
      database: AQI
      url: http://influxdb-service:8086
      isDefault: false
      editable: true

  - name: job3
      type: influxdb
      access: proxy
      database: cluster
      url: http://influxdb-service:8086
      isDefault: true
      editable: true
```

3. Create a `cluster.json` that is basically dashboard template of grafana and need to stored in the location `grafana-provisioning/dashboards/cluster.json` and template is here https://gitlab.cs.ut.ee/devops22fallpub/lab11-data-intensive. It is cluster.json.
4. Now run the code with a commit message as “job3-1 data processing tasks added.”
5. Now visualize the grafana dashboards by selecting **cluster dashboards** and visualize the clustered graphs.

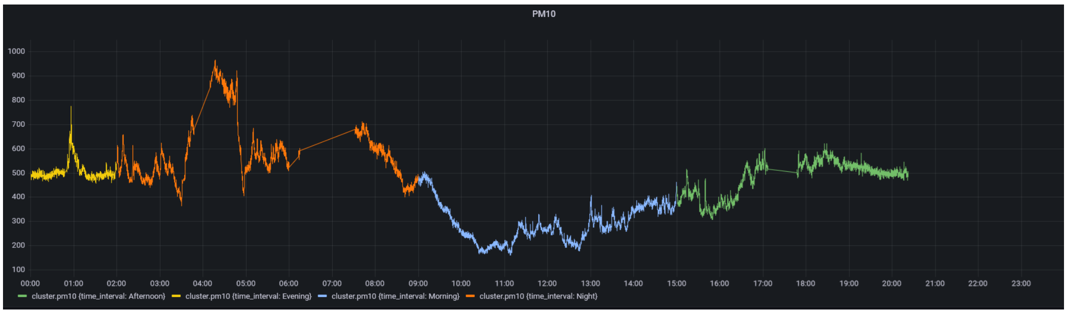
   PM2_5 day wise values


PM2_5
900
800
700
600
500
400
300
200
100
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
cluster.pm2_5 {time_interval: Afternoon}
cluster.pm2_5 {time_interval: Evening}
cluster.pm2_5 {time_interval: Morning}
cluster.pm2_5 {time_interval: Night}

Screenshot - 7

Take a screenshot of a webpage and the IP address should be clearly visible.

PM10 day wise values



Screenshot - 8

Take a screenshot of a webpage and the IP address should be clearly visible.

Deliverables

1- Gather all the screenshots

- Screenshot 1
- Screenshot 2
- Screenshot 3
- Screenshot 4
- Screenshot 5
- Screenshot 6
- Screenshot 7
- Screenshot 8

2- Download code of your GitLab project

3- Zip the code, screenshot and Upload the zip file to the course wiki page.

Don't delete your VMs

Remember that, we may see the commit history and the VMs while grading your submission.